EECS 280 Winter 2022 Final Exam

Fengyuan Hu (hufy)

This exam is randomized. Every student will receive a different set of questions chosen from a larger question bank. Individual questions found to be significantly more difficult than others will be curved in your favor. This exam also includes other unspecified countermeasures to prevent cheating.

Instructions

- **DO NOT** open this exam in more than one browser window.
- You may work on the questions in **any order**.
- **You must download and turn in your answers file to Canvas before exam time is up.**
- Read questions carefully, noting any **requirements or restrictions** for your solution.
- There is **no penalty for wrong answers**.
- It is to your benefit to record an answer to each question, even if you're not sure what the correct answer is.
- Assume all code is in **standard C++11**, and use only standard C++11 in your solutions.
- Throughout the exam, assume all necessary `#include` headers and the `using namespace std;` directive are present unless otherwise directed.
- You do not need to verify `REQUIRES` clauses with `assert` unless specifically instructed to do so.
- In code examples, `...` may be used to indicate code has been omitted for brevity. You may assume all omitted code works correctly and generally behaves as expected.

Questions and Reference Material Some sections include reference material with additional information and/or examples, shown on the right side of the page. Make sure to read this thoroughly for each question, and note that **you may have to scroll down to see all of the reference material**.

Allowed Resources

- You may use any notes or other resources, including online resources.
- You may use a compiler, IDE, or other programming tools.
- You may use the "Lobster" C++ tool, but be aware that it is not 100% C++ compliant and may compile/run incorrectly in some complex cases.

Prohibited Resources

- You are NOT allowed to use AI-assisted code writing tools such as GitHub Copilot. We have also tried to design exam questions for which these tools are not helpful.

Collaboration/Assistance is Prohibited

- You are NOT allowed to collaborate with anyone else to answer these questions.
- You are NOT allowed to solicit assistance from anyone to answer these questions.
- You are NOT allowed to screenshot, photograph, or otherwise reproduce these exam materials.

For example, reading existing posts on stackoverflow.com is OK, but posting new questions based on these exam materials to solicit help from others is prohibited.

This exam is given under the University of Michigan College of Engineering Honor Code.

In this problem, we will consider a container ADT that maintains a collection of stamps, where each stamp has a different country of origin, which is represented as a `string`. The class definition is provided in the reference material.

1.1 6 points ⭐

We would like to overload 3 operators. Based on the description of each, type the function **signature** (not implementation) needed for the operators. You may assume any operator overloads defined as member functions are defined within the class definition, so you don't need to worry about the scope resolution operator (`::`). You should pass objects by reference when appropriate, following the guidelines given in class. You should also mark parameters as `const` where appropriate.

The `[]` operator should take a string as input and return a vector of the collected stamps whose country of origin starts with that string. E.g.

```
StampCollection c1;
c1.add_stamp("Canada");
c1.add_stamp("Cambodia");
c1.add_stamp("China");
// currently 3 stamps in collection

vector<string> ca_stamps = c1["Ca"];
// ca_stamps -> { "Canada", "Cambodia" }
```

Fill in the function signature:

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

1: Containers 16 points

```
vector<string>  operator[](
const string &name ) const;
```

The `+=` operator should take a second `StampCollection` and merge it into the first, dropping any duplicates. It should support chaining multiple operations on one line. E.g.

```
// continuing from code above
StampCollection c2;
c2.add_stamp("US");
(c1 += c2).add_stamp("Laos");
// c1 now has 5 stamps, c2 still has 1 stamp
```

Fill in the function signature:

```
// Note: Implemented as a member function overload, within the StampCollection Class
StampCollection &  operator+=(
const StampCollection &other );
```

The `<<` operator should work when writing a `StampCollection` object to an `ostream` object and should print the number of stamps in the collection followed by a comma-separated list of the countries of origin. The operator should support multiple insertion operators (`<<`) on one line. E.g.

```
cout << c2 << endl; // prints "1-stamp collection: US" followed by newline to standard output
```

Fill in the function signature:

```
// Note: Implemented as a non-member function overload, outside the StampCollection Class
ostream &  operator<<(ostream& os,
const StampCollection &sc );
```

1.2 3 points

Consider the `get_first_stamp()` function, which returns the **first** (alphabetically) stamp country of origin from the member variable `set<string> collection`.

You may assume the `set` class in the standard library is implemented using the `BinarySearchTree` class you implemented in project 5, along with its `Iterator` class. Its data representation contains a single pointer to the root node of the underlying tree. You may assume the tree is always balanced, meaning for any node, there are approximately the same number of nodes in its left and right subtrees. The `<` operator is used for comparing elements.

If this function is implemented by performing the most efficient search for the appropriate stamp, how does the number of nodes accessed by `get_first_stamp()` grow with the number of elements in the `set` (i.e. the number of stamps in the collection)?

Put an X next to **exactly one** choice to indicate your answer.

- [ ] O(1) - Does not depend on size
- [X] O(logN) - Grows logarithmically with size
- [ ] O(N) - Grows linearly proportional to size
- [ ] O(N^2) - Grows proportional to square of size

In 1-2 sentences, briefly justify your answer.

Since the set is using a BST to store its data, the get_first_stamp is just finding the leftmost element in the tree. For a balanced BST tree of height n, it has around 2^n elements and finding the leftmost element takes n steps, which is log2(2^n).

1.3 3 points

Consider the `sell_excess_stamps_from()` function. You can assume that vectors store their data contiguously in memory as an unsorted array. If implemented optimally, how does the number of vector elements accessed by `sell_excess_stamps_from()` grow with the size of the `for_sale` vector?

Reference Material (Section 1) 40%

```
class StampCollection {
private:

  // Set of unique stamps in your collection
  // stamp represented by a string corresponding to country
  // of origin
  // A set does not allow duplicates
  set<string> collection;

  // Unsorted vector of excess stamps that you want to sell
  vector<string> for_sale;

public:
  // EFFECTS: If no stamp in collection matches "country",
  //          add it to collection.
  //          Otherwise, add "country" to for_sale
  void add_stamp(const string& country);

  // EFFECTS: Returns the first stamp (sorted alphabetically by
  //          country of origin) in your collection
  string get_first_stamp() const;

  // EFFECTS: Removes all stamps from a particular country
  //          from for_sale vector.
  void sell_excess_stamps_from(const string& country);

};
```

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

Hide **Time Elapsed**
1h 48m 10s
This is not an official timer.
Please submit your answers file before the deadline.

100 points

16 pts 1: Containers
6 pts Question 1.1
4 pts 2: Templates
24 pts 3: Dynamic Memor...
10 pts 4: Linked Lists
10 pts Question 4.1
18 pts 5: Iterators and Fu...
5 pts 6: Types of Recursi...
21 pts 7: Recursion on Tre...
2 pts 8: Thanks!

Put an X next to **exactly one** choice to indicate your answer.

	O(1) - Does not depend on size
	O(logN) - Grows logarithmically with size
X	O(N) - Grows linearly proportional to size
	O(N^2) - Grows proportional to square of size

In 1-2 sentences, briefly justify your answer.

Since the for_sale vector is unsorted and is stored contiguously, to find all the stamps of a specific country we have to access all the elements in the vector, which is a linear operation. Thus the number of elements accessed by sell_excess_stamps_from() grow linearly with the size of the for_sale vector.

1.4 2 points

Assume that `StampCollection` has been implemented as specified (including the Big 3), and that it has followed the guidelines laid out in class that memory resources are obtained automatically during initialization of a `StampCollection`, and are released when the object is destroyed.

Does the following code leak memory?

```
int main() {
  StampCollection* sc = new StampCollection;
  string* s = new string("EECSlandia");
  sc->add_stamp(*s);
  delete sc;
}
```

- (●) Yes - the code does leak memory
- ( ) No - the code does not leak memory

1.5 2 points

Assume that `StampCollection` has been implemented as specified (including the Big 3), and that it has followed the guidelines laid out in class that memory resources are obtained automatically during initialization of a `StampCollection`, and are released when the object is destroyed.

Does the following code leak memory?

```
int main() {
  StampCollection* sc = new StampCollection;
  sc->add_stamp("U.S.");
  delete sc;
}
```

- ( ) Yes - the code does leak memory
- (●) No - the code does not leak memory

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

2: Templates 4 points

Reference Material (Section 2) 40%

2.1 4 points

Select each of the following instances where it would be apprioriate to use a template.

- [ ] Implementing a hierarchy of classes with different versions of a function, where the dynamic type of the calling object determines which function version is called
- [x] Implementing one container class that holds any single type of data
- [x] Writing a function that should work for all types of input that have the `++` operator
- [ ] Implementing a container that stores days of the month as integers
- [x] Implementing a container that holds any single type of data as a circular array
- [ ] Writing a function that flips a coin and returns `'H'` or `'T'`
- [ ] Implementing a `Team` class that stores four Euchre players but gets user input to determine how many are `SimplePlayer` and how many are `HumanPlayer`

3: Dynamic Memory and The Big Three 24 points

Reference Material (Section 3) 40%

Consider the classes shown in the reference material used to represent a `Library`, which contains a dynamically allocated array of `Book`s that it owns. Make sure to take a look at the **code and comments in the reference material**, which contain additional details on the structure of the classes and their member variables.

Note that the `Library` has a member variable `books_begin`, which points to the beginning of the dynamically allocated array, as well as a member variable `books_end`, which points to the memory location one past the last valid `Book` element in the array.

3.1 6 points

Implement a default constructor for `Library` as it would appear **inside** the class defininition.

The constructor should initialize the object such that its dynamically allocated array has a capacity of `100` but does not yet contain any valid `Book` elements.

All member variables should be initialized appropriately and your implementation *must* use a member initializer list where appropriate.

Theme: Light Dark None

```
Library() : books_begin(new Book[100]), books_end(books_begin), books_capacity
}
```

```
class Book {
private:
  int num_copies;
  string title;
public:

  // Default ctor
  Book()
   : num_copies(0), title("") { }

  // Custom ctor
  Book(int num_copies_in, const string &title_in)
   : num_copies(num_copies_in), title(title_in) { }
};

class Library {
private:
  Book * books_begin; // dynamic array of books
  // books_end points one past the last book
  // or to the beginning of the array if empty
  Book * books_end;
  int books_capacity; // length of dynamic array

public:

  int size() const {
    return books_end - books_begin;
  }

  ~Library() {
    delete[] books_begin;
  }

  ...

};
```

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

3.2 3 points

Consider the following RME and implementation code for a new member function, `remove_back_book()`. Is the code correct, working as intended for all cases?

If the code is correct, put an "X" in the small box next to "correct" and leave the large justification box blank.

If the code is not correct, put an "X" in **one** of the small boxes to indicate the kind of error that could occur. Then, in the large box, write a brief justification. For example, you might explain how the code directly causes an error by dereferencing a null pointer or how the function contributes to a memory leak. Or, you might describe how the function messes up the data structure and give a specific future scenario where this would manifest in an error. You may be brief, but make sure to be specific.

Do not choose more than one kind of error. There may be multiple possible right answers for some questions, but choose **only one**.

```
// REQUIRES: There is at least one Book in this Library
//           For your analysis in this question, you MAY assume the function is
//           never called in a way that violates this requires clause.
// EFFECTS:  Removes the Book at the back of the dynamically
//           allocated array.
void Library::remove_back_book() {
  --books_end;
  delete books_end;
}
```

- [ ] correct
- [ ] null pointer dereference
- [X] memory leak
- [ ] double delete (double free)
- [ ] delete a non-heap object
- [ ] use of dead object
- [ ] out-of-bounds access
- [ ] use of uninitialized object

Justification (leave blank if the code is correct)

When the library didn't reach its capacity, meaning that there are other Book objects after the Book at the "back", removing the Book at the back will break the dynamic array and cause the Book objects after the Book at the back to leak memory.

3.3 3 points

Consider the following RME and implementation code for a new member function, `add_book()`. Is the code correct, working as intended for all cases?

If the code is correct, put an "X" in the small box next to "correct" and leave the large justification box blank.

If the code is not correct, put an "X" in **one** of the small boxes to indicate the kind of error that could occur. Then, in the large box, write a brief justification. For example, you might explain how the code directly causes an error by dereferencing a null pointer or how the function contributes to a memory leak. Or, you might describe how the function messes up the data structure and give a specific future scenario where this would manifest in an error. You may be brief, but make sure to be specific.

Do not choose more than one kind of error. There may be multiple possible right answers for some questions, but choose **only one**.

```
// REQUIRES: There is enough space in the dynamic array to add another book.
//           For your analysis in this question, you MAY assume the function is
//           never called in a way that violates this requires clause.
// EFFECTS:  Adds the given book to the back of the dynamically allocated
array.
void Library::add_book(const Book &to_add) {
  Book *temp = new Book(to_add);
  *books_end = *temp;
  ++books_end;
}
```

- [ ] correct
- [ ] null pointer dereference
- [X] memory leak

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

Hide **Time Elapsed**
1h 48m 10s
This is not an official timer.
Please submit your answers file before the deadline.

100 points

16 pts 1: Containers
6 pts Question 1.1
4 pts 2: Templates
24 pts 3: Dynamic Memor...
10 pts 4: Linked Lists
10 pts Question 4.1
18 pts 5: Iterators and Fu...
5 pts 6: Types of Recursi...
21 pts 7: Recursion on Tre...
2 pts 8: Thanks!

- [ ] double delete (double free)
- [ ] delete a non-heap object
- [ ] use of dead object
- [ ] out-of-bounds access
- [ ] use of uninitialized object

Justification (leave blank if the code is correct)

After *books_end = *temp, the Book that temp points to was copied to the dynamic array, and it should be deleted. Since it's not deleted and temp goes out of scope after the function returns, we lost track of the Book created in the function and this is a memory leak.

3.4 10 points

Implement the `Library` assignment operator as it would be defined inside the `Library` class by filling in the boxes below.

If you believe a blank/box should be **empty**, simply leave it blank.

When finished, your implementation should perform a **deep copy** where appropriate and **avoid undefined behavior or memory leaks**.

```
// Assignment operator for the Library class
Library &  operator=( const Library &  rhs)
           {

  // Check for self assignment and respond appropriately
  if (this == &rhs) {
    return *this;
  }

  // Delete old data for this Library as appropriate
  delete[] books_begin;

  // Copy data, making a deep copy where appropriate
  // After the copy, this Library should have the same capacity as rhs
  books_begin =
new Book[rhs.books_capacity] ;
  Book * ptr = books_begin;
  Book * rhs_ptr = rhs.books_begin;
  while(rhs_ptr <
rhs.books_end ) {
    *ptr = *rhs_ptr ;
    ++ptr;
    ++rhs_ptr;
  }
  books_end = ptr ;
  books_capacity =
rhs.books_capacity ;

  // Any cleanup work needed after copying data
  // DO NOT COPY MEMBER VARIABLES HERE


  return *this ;

}
```

3.5 2 points

Consider a new class `Country` with this data representation:

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

Hide **Time Elapsed**
1h 48m 10s
This is not an official timer.
Please submit your answers file before the deadline.

100 points

16 pts 1: Containers
6 pts Question 1.1
4 pts 2: Templates
24 pts 3: Dynamic Memor...
10 pts 4: Linked Lists
10 pts Question 4.1
18 pts 5: Iterators and Fu...
5 pts 6: Types of Recursi...
21 pts 7: Recursion on Tre...
2 pts 8: Thanks!

```
class Country {
private:

  // A pointer to the oldest library
  // within the Country
  Library * oldest_library;

public:
  ...
};
```

Does the `Country` class need custom implementations of the Big Three? (You may

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

4: Linked Lists 10 points

4.1 10 points

In this question, implement a member function `populate()` of the sorted, singly-linked list of integers shown in the reference material. You may not call any member functions of the SortedIntList in your solution.

Reference Material (Section 4) 40%

SortedIntList `class`:

```
class SortedIntList {
public:
  SortedIntList() : first(nullptr), size(0) { };
  void populate();
  // other member functions not shown
```

Hide **Time Elapsed**
1h 48m 10s
This is not an official timer.
Please submit your answers file before the deadline.

100 points

16 pts 1: Containers
6 pts Question 1.1
4 pts 2: Templates
24 pts 3: Dynamic Memor...
10 pts 4: Linked Lists
10 pts Question 4.1
18 pts 5: Iterators and Fu...
5 pts 6: Types of Recursi...
21 pts 7: Recursion on Tre...
2 pts 8: Thanks!

Fill in the blanks below. If you believe a blank/box should be **empty**, simply leave it blank.

```
// REQUIRES: SortedIntList is sorted from smallest to largest with no
duplicates
// MODIFIES: *this
// EFFECTS:  Inserts elements as needed into the list such that each element
//           has a value one greater than the previous element. The first and
//           last values in the list remain the same. Empty lists are not
modified.
//           EXAMPLES:
//           {1->3->6} yields {1->2->3->4->5->6}
//           {4} yields {4}
void SortedIntList::populate() {

  if (!first) {
    return;
  }

  int num = first -> datum;
  Node *ptr_1 = first;
  Node *ptr_2 = first -> next;

  while (
!ptr_2
) {
    num++;
    while (num < ptr_2->datum) {
      // insert new node
      // Remember this is a **singly** linked list
      Node *temp = new Node;
      temp->datum = num;
      temp->next = ptr_2;
      ptr_1->next = temp;
      ptr_1 = temp;

      num++;
    }
    ptr_1 = ptr_1->next;
    ptr_2 = ptr_2->next;
  }
  return;

}
```

```
private:
  // INVARIANT: Nodes are maintained such that
  // their elements are in ascending sorted order
  // and there are no duplicates

  struct Node {
    int datum;
    Node *next;
  };

  Node *first;
  int size;
};
```

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

5: Iterators and Functors 18 points

You've just been hired as a fashion intern for the reality TV show Queer Eye, which features a team of professionals (the Fab 5) giving lifestyle and fashion makeovers. Working under fashion designer Tan France, you've been tasked with purchasing shoes for the next person to receive the Fab 5 makeover. Luckily, you are able to apply your skills from EECS 280 to help you optimize this task. You decide to write the code below to help you purchase shoes that will fit the individual receiving the makeover.

5.1 8 points

Implement the predicate functor `ShoeFits` below. The constructor should initialize the individual's shoe size. **Use an initializer list if possible**. You will also need to implement the `operator()` so that given a `Shoe` object, the functor evaluates whether or not the shoe will 'fit'. See the **reference material** for the definition of the `Shoe` struct and for examples.

Note that shoe sizes only range from 0 to 22 inclusive and increase in 0.5 increments. You do not need to worry about numeric precision or roundoff error when comparing doubles.

You may find the following `abs` function helpful:

```
// EFFECTS: Returns the absolute value of parameter n.
double abs(double n);
```

If you believe a blank/box should be **empty**, simply leave it blank.

Reference Material (Section 5) 40%

Shoe `struct`:

```
struct Shoe {
  // size is either a whole number or a half size (e.g. 9.5)
  double size;
  bool is_comfy;
};
```

You may use this `abs` function as a helper:

```
// EFFECTS: Returns the absolute value of parameter n.
double abs(double n);
```

Example for `ShoeFits` functor:

```
Shoe s1 = {9, true};
Shoe s2 = {9.5, true};
Shoe s3 = {10, false};
Shoe s4 = {10.5, false};

ShoeFits f(10);

cout << f(s1) << endl; // prints '0' (false)
cout << f(s2) << endl; // prints '1' (true)
cout << f(s3) << endl; // prints '1' (true)
cout << f(s4) << endl; // prints '0' (false)
```

Example for `select_shoes`:

```
vector<Shoe> shoeStore;
shoeStore.push_back({9, true});
shoeStore.push_back({9.5, true});
shoeStore.push_back({10, false});
shoeStore.push_back({10.5, false});

vector<Shoe> shoppingCart(shoeStore.size());
select_shoes(shoeStore, shoppingCart, 10);

// shoppingCart now contains { {9.5, true}, {10, false} }
```

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

Hide **Time Elapsed**
1h 48m 10s
This is not an official timer.
Please submit your answers file before the deadline.

100 points

16 pts 1: Containers
6 pts Question 1.1
4 pts 2: Templates
24 pts 3: Dynamic Memor...
10 pts 4: Linked Lists
10 pts Question 4.1
18 pts 5: Iterators and Fu...
5 pts 6: Types of Recursi...
21 pts 7: Recursion on Tre...
2 pts 8: Thanks!

```
class ShoeFits {
private:
  // The individual's shoe size
  double size;

public:
  // Constructor
  ShoeFits(
int size_in
)
    : size(size_in)
{

  }

  // EFFECTS: Returns whether or not a shoe fits. Specifically, a shoe fits
  //          when the shoe size either (1) matches the individual's size
exactly
  //          OR (2) is within 0.5 of the individual's size (inclusive) and
  //          is_comfy is true.
  //          Otherwise, the shoe does not fit.
   bool operator() ( const Shoe & shoe) const
{
    if( shoe.size == size ) {
      // Matches the individual's shoe size exactly
      return true;
    }
    else if(
abs(shoe.size - size) <= 0.5 && shoe.is_comfy
) {
      // within 0.5 of the individual's size (inclusive) and the shoe is comfy
      return true;
    }
     return false ;
  }
};
```

5.2 3 points

Consider the **buggy** implementation of `copy_if` below. The function should copy elements from the container starting at `begin` (and up to but not including `end`) into the container starting at `dest`. If there are values already in `dest` they will be overwritten. Only elements for which the predicate `pred` returns true should be copied.

Note: Use an example to help you trace through the code. If you are having a hard time identifying the bug, consider skipping this question and returning to it later.

```
// REQUIRES: begin, end, and dest are valid Iterators.
//           begin is before or equal to end.
//           The container starting at dest has sufficient
//           space to hold copied elements.
// MODIFIES: *dest
// EFFECTS: Copies elements to container starting at dest
//          from container starting at begin
template <typename InputIter, typename OutputIter, typename Pred>
void copy_if(InputIter begin, InputIter end, OutputIter dest, Pred pred) {
  for(; *begin != *end; ++begin) {
    if(pred(*begin)) {
      *dest = *begin;
      ++dest;
    }
  }
}
```

Describe the bug in `copy_if` and explain why it is incorrect.

*begin != *end this is incorrect. Firstly end may not be dereferenceable, and second we should directly compare the iterators instead of comparing the data the iterators are pointing to.

5.3 3 points

Describe something you can do with a functor that you can't do with a function pointer.

We can use runtime data to initialize functors but we can't do the same with function pointers.

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

Hide **Time Elapsed**
1h 48m 10s
This is not an official timer.
Please submit your answers file before the deadline.

100 points

16 pts 1: Containers
6 pts Question 1.1
4 pts 2: Templates
24 pts 3: Dynamic Memor...
10 pts 4: Linked Lists
10 pts Question 4.1
18 pts 5: Iterators and Fu...
5 pts 6: Types of Recursi...
21 pts 7: Recursion on Tre...
2 pts 8: Thanks!

5.4 4 points

Now use `copy_if` and `ShoeFits` to implement the `select_shoes` function according to its RME. The function should copy `Shoes` that fit for the given individual's `size` from `shoeStore` into `shoppingCart`.

- You **MUST** use both `copy_if` and `ShoeFits`.
- You may **NOT** use any vector operations except `begin()` and `end()`.
- You may **NOT** use loops or recursion.

For convenience, we have copied the signature for `copy_if` here. Assume that it works **correctly** (i.e. the same RME and signature as above, but not the buggy implementation).

```
template <typename InputIter, typename OutputIter, typename Pred>
void copy_if(InputIter begin, InputIter end, OutputIter dest, Pred pred)
```

Theme: Light | Dark | None

```
// REQUIRES: shoppingCart has sufficient space to hold all copied shoes
// MODIFIES: shoppingCart
// EFFECTS: Copies Shoes from shoeStore into shoppingCart if the given
//          Shoe fits according to the individual's size (and, where
//          applicable, whether the shoe is comfy).
void select_shoes(const vector<Shoe> &shoeStore, vector<Shoe> &shoppingCart,
double size) {
  ShoeFits sf(size);
  copy_if(shoeStore.begin(), shoeStore.end(), shoppingCart.begin(), sf);
}
```

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

6: Types of Recursion 5 points

Reference Material (Section 6) 40%

This section has no reference material.

In the questions below, determine the type of recursion used by each function.

(**Please note** that due to randomization, it is possible you may happen to get several questions with the same answer. Do your best to consider each question below individually.)

6.1 2.5 points

Consider the function definition below and select the best answer choice to describe the kind of recursion used in the function's implementation.

```
int fn(int x, int y) {
  if (x == 0 || y == 0) {
    return 0;
  }
  else if (x < y) {
    return x + fn(x, y-1);
  }
  else {
    return y + fn(x-1, y);
  }
}
```

- ○ Tail recursive
- ◉ Linear recursive (non-tail)
- ○ Tree recursive
- ○ Not recursive

6.2 2.5 points

Consider the function definition below and select the best answer choice to describe the kind of recursion used in the function's implementation.

```
struct Node {
  int datum;
  Node *left;
  Node *right;
};

int fn(const Node *n) {
  if (!n) {
    return 0;
  }
  if (n->datum < 0) {
    return 1 + fn(n->left);
  }
  else {
    return 1 + fn(n->right);
  }
}
```

- ○ Tail recursive
- ◉ Linear recursive (non-tail)
- ○ Tree recursive
- ○ Not recursive

7: Recursion on Trees 21 points

In this section, you will work with a structure of nodes representing a **binary tree (*not necessarily a binary search tree*)**, which may contain duplicate elements. See the **reference material** for details.

Both questions in this section deal with distances to certain nodes in a tree. To represent the possibility that a node was "not found", they use a very large integer constant `MAX` that is larger than any reasonable distance within a tree (but not so large that numeric overflow is a concern). If a function returns a value >= MAX, it means it did not find its target node/nodes. This note is repeated in the reference material.

7.1 9 points

Now that the clownfish Nemo is all grown up, he is excited to make his path back from his home in the Great Barrier Reef to visit his old haunt in Sydney. He has to figure out which ocean currents to ride to swim the shortest route to Sydney Harbor. These are represented as a tree, with examples shown in the reference material.

Note that for example, "Sydney" may occur in a tree multiple times because there are multiple routes to get to Sydney.

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

Reference Material (Section 7) 40%

Nodes and Tree Structure

```
struct Node {
  string node_kind; // Could be "Reef" or
specific harbor
  Node *left;
  Node *right;
};
// a null pointer represents
// an empty tree
```

`MAX` Constant

Both questions in this section deal with distances to certain nodes in a tree. To represent the possibility that a node was "not found", they use a very large integer constant `MAX` that is larger than any reasonable distance within a tree (but not so large that numeric overflow is a concern). If a function returns a value >= MAX, it means it did not find its target node/nodes.

```
const int MAX = /* a very large number */;
```

Implement the `dash_to_harbor()` function, which returns the length of the shortest path to a specific harbor in a given tree.

See the **reference material** for additional information and examples.

- Your implementation should work correctly for any binary tree.
- Your code **must** use recursion (and **may not** use any loops).
- You can assume that any harbor in the tree will be at a leaf node.

You may use this helper function in your code:

```
// EFFECTS: Returns the lesser of x and y.
int min(int x, int y);
```

Theme: Light Dark None

```
// REQUIRES: node points to a valid binary tree of nodes. Nodes may represent
// a harbor or a reef, though a harbor may only occur at a leaf node.
// EFFECTS: Returns the length of the shortest path from the root node to any
node
// matching the given harbor. If no such node is found, returns a value >= MAX.
int dash_to_harbor(const Node *node, const string &harbor) {
  if (!node) {
    return MAX;
  }
  if (node->node_kind == harbor) {
    return 0;
  }
  return 1 + min(dash_to_harbor(node->left, harbor), dash_to_harbor(node->right,
  harbor));



}
```

`dash_to_harbor()` Examples

- The nodes in a tree could contain reefs or harbors.
- Only leaf nodes may contain harbors.
- Distance considered moving down (no going up then back down).
- An **empty** tree has a shortest path of length MAX.

```
    Reef  = root
   /    \
 Reef    Reef
   \      \
   Reef    Sydney
   /  \
Sydney  Reef
```

```
// Example: In the tree above,
dash_to_harbor(root, "Sydney")
// returns 2, the length of the shortest path to
Sydney.
```

7.2 12 points

Nemo and his turtle buddies embark on a trip to harbors around Sydney. Their quest is to find the reef that is closest to all chosen harbors in the Pacific. The `nearest_reef_to_harbors()` function, finds the node that has the least distance to all of the specified harbors and returns a pair comprising of that node and the least distance. Only a leaf node can be a habor (not all leaf nodes are harbors).

See the **reference material** for examples.

Implement `nearest_reef_to_harbors()` by filling in the blanks below.

You must use the function `dash_to_harbor()` from the first part as a helper to implement `nearest_reef_to_harbors()`, and you may assume it works correctly according to its RME.

`nearest_reef_to_harbors()` Examples

```
    Reef  = root
   /    \
 Reef    Reef
 /      /   \
Reef  Reef  Sydney
  \     \
 Sydney  Osaka
```

```
// Example 1. In the tree above, the following
call returns a pair containing
// a pointer to the bolded red node and the value
3 (since it's
// 1 away from Sydney and 2 away from Osaka,
1+2=3)
vector<string> harbors_to_visit = { "Sydney",
"Osaka" };
std::pair<Node*, int> p =
nearest_reef_to_harbors(root, harbors_to_visit);

// Example 2. In the same tree above, the
following call using the
// changed vector returns a pair containing a
pointer to the root node
// and a value >= MAX (since not all the harbors
could be found).
vector<string> harbors_to_visit = { "Sydney",
```

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

Hide **Time Elapsed**
1h 48m 10s
This is not an official timer.
Please submit your answers file before the deadline.

100 points

16 pts 1: Containers
6 pts Question 1.1
4 pts 2: Templates
24 pts 3: Dynamic Memor...
10 pts 4: Linked Lists
10 pts Question 4.1
18 pts 5: Iterators and Fu...
5 pts 6: Types of Recursi...
21 pts 7: Recursion on Tre...
2 pts 8: Thanks!

```
// REQUIRES: node points to a valid binary tree of nodes.
// REQUIRES: harbors_to_visit is not empty and does not contain duplicates.
// EFFECTS: Returns a std::pair<Node*, int> that contains a pointer to the Node
that has
// the smallest total distance to ALL of the harbors specified in the
harbors_to_visit
// vector as the first element and that total distance as the second element of
the
// pair. If no such result is found, returns a pointer to the root and a value
>= MAX.
std::pair <Node*,int> nearest_reef_to_harbors(Node *node, const vector<string>
&harbors_to_visit) {

  // Base case, empty tree
  if ( !node ) {
    std::pair <Node*,int> p(node, MAX );
    return p;
  }

  // Make recursive calls
  std::pair <Node*,int> left =
nearest_reef_to_harbors(node->left, harbors_to_visit) ;
  std::pair <Node*,int> right =
nearest_reef_to_harbors(node->right, harbors_to_visit) ;

  // Determine whether we need to check distance to harbors from this node
  // or whether one of the recursive results should just be used instead.
  if (left.second >= MAX && right.second >=
MAX ) {
    int dist = 0;
    for(int i = 0; i < harbors_to_visit.size(); ++i){
      dist += dash_to_harbor(node, harbors_to_visit[i]) ;
    }
    std::pair <Node*,int> p( node ,
dist );
    return p;
  }
  else if ( right.second >= MAX ) {
    return left;
  }
  else {
    return right;
  }

}
```

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.

8: Thanks! 2 points

8.1 2 points

Thank you for the effort and care you put into 280! Well done! We know it's a challenging course and that it's a lot of work, but we hope you got a lot out of it and had a chance to grow.

We are proud of all you have accomplished, and we wish you the best in the future!

We ended up with 2 extra points, so please enjoy some free points on us!

- (●) Yes, I would like the free points.
- ( ) No, I would not like the free points.

Reference Material (Section 8) 40%

This section has no reference material.

You've reached the bottom of the exam! If you're done, make sure to click the **"Answers File"** button, download a **`.json` answers file**, and submit it before the end of the exam!

If you have **questions during the exam**, you may post on Piazza or join our Zoom

Answers File

Download an answers file to submit separately.